UNIVERSIDAD AUTONOMA DE SAN LUIS POTOSI

FACULTAD DE INGENIERÍA

Fundamentos de desarrollo móvil

José Luis Medina Delgado

Ximena Guadalupe Rebolloso Rodríguez

Manual del programador: Plantly

A 5 de diciembre de 2022, San Luis Potosí, S.L.P.

Manual del programador

Introducción

En este anexo se describe la documentación técnica de programación, incluyendo la instalación del entorno de desarrollo, la estructura de la aplicación, su compilación, la configuración de los diferentes servicios de integración utilizados o las baterías de test realizadas.

Estructura de directorios

El repositorio del proyecto se distribuye de la siguiente manera:

`/`: contiene los ficheros de configuración, el fichero README y la copia de la licencia.

prueba0/: módulo correspondiente a la aplicación.

lib\main.dart: contiene el acceso principal dentro de la app

prueba0\lib\page: Contiene los dart que compone la aplicación y que se mandan a llamar entre sí.

lib\models: Contiene los modelos y estructuras de

lib\services\firebase_crud.dart: modulo correspondiente a los servicios de firebase para las funcionalidades del crud.

Lib\test: test unitarios.

Manual del programador

El siguiente manual tiene como objetivo servir de referencia a futuros programadores que trabajen en la aplicación. En él se explica cómo montar el entorno de desarrollo, obtener el código fuente del proyecto, compilarlo, ejecutarlo, testearlo y exportarlo.

Entorno de desarrollo

Para trabajar con el proyecto se necesita tener instalados los siguientes programas y dependencias:

Flutter.

Android Studio.

Git.

Firebase library.

Modelos

Modelo/ Employee

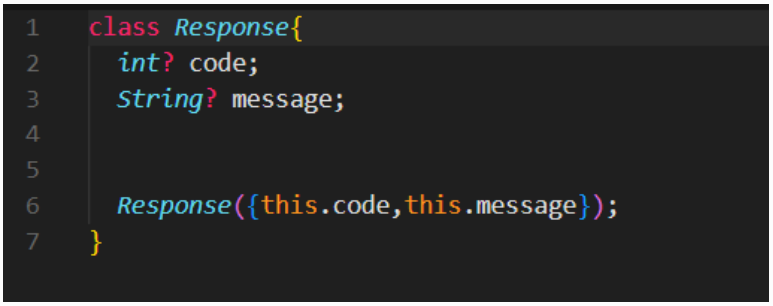
```dart
import 'dart:ffi';

class Plant{
  String? uid;
  String? plantanombre;
  String? plantadescripcion;
  String? platacondicion;
  String? plantaexposicion;
  String? recordatorio;

  Plant({this.uid,this.plantanombre,this.plantadescripcion,this.platacondicion,this.plantaexposicion,this.recordatorio});
}
```

El modelo consiste en 6 variables tipo string donde se almacenan y guardan la información necesaria para la colección de ¨Plants¨ que se guardaran en una base datos alojada en firebase.

Modelo/ Response



```dart
class Response{
  int? code;
  String? message;


  Response({this.code,this.message});
}
```

El modelo consiste en 2 variables tipo int y string respectivamente donde se almacenan los mensajes que se mandaran a llamar cuando era requerido dentro del Firebease service.

Modelo/ User

```dart
class User{
  String? userid;
  String? username;
  String? userpassword;
  String? useremail;

  User({this.userid,this.username,this.userpassword,this.useremail});
}
```

El modelo consiste en 4 variables tipo string donde se almacenan y guardan la información necesaria para la colección de ¨User¨ que se guardaran en una base datos alojada en firebase

Paginas

Main

```dart
import 'package:prueba0/page/addpage.dart';
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:prueba0/page/loginpage.dart';
import 'firebase_options.dart';

Run | Debug | Profile
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(options: DefaultFirebaseOptions.currentPlatform,);
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Plantly',
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        primarySwatch: Colors.indigo,
        brightness: Brightness.dark,
      ), // ThemeData
      //home: AddPage(),
      home: IniciarSesion(),
    ); // MaterialApp
  }
}
```

La primera sección se encarga de mandar a llamar las diferentes rutas, y paquetes necesarios para que funcione correctamente, la función del main se redirigir a la página IniciarSesion , llamando la función principal.

Servicie Firebase

```dart
import 'package:cloud_firestore/cloud_firestore.dart';
import '../models/response.dart';

final FirebaseFirestore _firestore = FirebaseFirestore.instance;
final CollectionReference _Collection = _firestore.collection('Plantas');

class FirebaseCrud {

  static Future<Response> addPlant({
    required String plantanombre,
    required String plantadescripcion,
    required String plantacondicion,
    required String plantarecordatorio,
    required String plantaexposicion,
  }) async {

    Response response = Response();
    DocumentReference documentReferencer =
        _Collection.doc();

    Map<String, dynamic> data = <String, dynamic>{
      "planta_nombre": plantanombre,
      "planta_descripcion": plantadescripcion,
      "planta_condicion" : plantacondicion,
      "planta_recordatorio": plantarecordatorio,
      "planta_exposicion": plantaexposicion,

    };
```

Se mandan a llamar los paquetes correspondientes, así como como el modelo response necesario, se crea una instancia de la base datos, así como la colección ¨plantas¨, la clase FirebaseCrud sirve para poder acceder a los datos y crear una estructura dinámica.

```
    var result = await documentReferencer
        .set(data)
        .whenComplete(() {
          response.code = 200;
          response.message = "Añadida correctamente";
        })
        .catchError((e) {
            response.code = 500;
            response.message = e;
        });

        return response;
  }

  static Future<Response> addUser({
   required String usuarionombre,
   required String usuarioemail,
   required String usuariocontras,

```

En esta sección se guarda la variable Result para poder mandar a llamar el mensaje de éxito o de error cuando sea necesario, en la siguiente sección se crea una estructura Future, con la que podamos usar los datos de usuario cuando sean llamados con addUser

```
  }) async {

    Response response = Response();
    DocumentReference documentReferencer =
        _Collection.doc();

    Map<String, dynamic> data = <String, dynamic>{
      "usuario_nombre": usuarionombre,
      "usuario_email": usuarioemail,
      "usuario_contras" : usuariocontras,


    };

    var result = await documentReferencer
        .set(data)
        .whenComplete(() {
          response.code = 200;
          response.message = "Añadida correctamente";
        })
        .catchError((e) {
            response.code = 500;
            response.message = e;
        });

        return response;
  }
```

Aquí podremos mandar a llamar la función Response, para signar ala variable response la referencia.

Continuamos con una estructura Dinámica, donde usamos asignamos los datos del usuario a su correspondiente caso, continuando tendremos la llamada de éxito o error en caso de que se agreguen los datos.

```dart
static Stream<QuerySnapshot> readPlant() {
  CollectionReference notesItemCollection =
      _Collection;

  return notesItemCollection.snapshots();
}

static Future<Response> deletePlant({
  required String docId,
}) async {
   Response response = Response();
  DocumentReference documentReferencer =
      _Collection.doc(docId);

  await documentReferencer
      .delete()
      .whenComplete((){
        response.code = 200;
        response.message = "Eliminada correctamente";
      })
      .catchError((e) {
        response.code = 500;
        response.message = e;
      });

 return response;
}
```

Finalmente, en la función creamos las funciones ReadPlant donde accedemos a la información de la base de datos por medio de un snapshots. Continuando con deleteplant, donde hacemos uso de documentReferences, para acceder a la base de datos y eliminarlos correctamente.

AddPage

```dart
import 'package:prueba0/page/listpage.dart';
import 'package:flutter/material.dart';
import '../services/firebase_crud.dart';

class AddPage extends StatefulWidget {
  @override
  State<StatefulWidget> createState() {
    Create Jira Issue
    // TODO: implement createState
    return _AddPage();
  }
}

class _AddPage extends State<AddPage> {
  final _planta_nombre = TextEditingController();
  final _planta_description = TextEditingController();
  final _planta_condicion = TextEditingController();
  final _planta_recordatorio = TextEditingController();
  final _planta_exposicion = TextEditingController();
  bool? checkboxListTileValue1;
  bool? checkboxListTileValue2;
  final GlobalKey<FormState> _formKey = GlobalKey<FormState>();
```

Se importan las librerías y rutas correspondientes, se crea la clase Addpage, y se crea un State, en el que se manda a llamar diferentes TextEditController con los que manipularemos los datos necesarios.

```dart
@override
Widget build(BuildContext context) {
  //Textform Nombre de planta
  final plantanombre = TextFormField(
      controller: _planta_nombre,
      autofocus: false,
      validator: (value) {
        if (value == null || value.trim().isEmpty) {
          return 'Este elemento es necesario';
        }
      }, //para poner color al border
      decoration: InputDecoration(
          enabledBorder: OutlineInputBorder(borderSide: BorderSide(color: Colors.black)),
          contentPadding: const EdgeInsets.fromLTRB(20.0, 15.0, 20.0, 15.0),
          hintText: "Nombre de la planta",
          border: OutlineInputBorder(),
          hintStyle: const TextStyle(color: Colors.black)), // InputDecoration
      style: const TextStyle(color: Colors.black),); // TextFormField
  //Texform descripcion de planta
```

Se crean los Final de los textFormField , que son los controladores para que guarden los datos, la estructura que tiene es para que tenga el estilo definido en los moodboard , la estructura se repite para los text de Nombre, Estado, Recordatorio Condición, Exposición.

```dart
  final viewListbutton = TextButton(
      onPressed: () {
        Navigator.pushAndRemoveUntil<dynamic>(
          context,
          MaterialPageRoute<dynamic>(
            builder: (BuildContext context) => ListPage(),
          ), // MaterialPageRoute
          (route) => false, //if you want to disable back feature set to false
        );
      },
      child: const Text('Veer lista de plantas')); // TextButton

  final SaveButon = Material(
    elevation: 5.0,
    borderRadius: BorderRadius.circular(30.0),
    color: const Color(0xFF9DBE8B),
    child: MaterialButton(
      minWidth: MediaQuery.of(context).size.width,
      padding: const EdgeInsets.fromLTRB(20.0, 15.0, 20.0, 15.0),
      onPressed: () async {
        if (_formKey.currentState!.validate()) {
          var response = await FirebaseCrud.addPlant(
              plantanombre: _planta_nombre.text,
              plantadescripcion: _planta_description.text,
              plantarecordatorio: _planta_recordatorio.text,
              plantacondicion: _planta_condicion.text,
              plantaexposicion: _planta_exposicion.text,
              );
          if (response.code != 200) {
```

Se crearon diferentes botones, como viewButton que servirá para dirigir a la página donde se desplegará la lista de plantas registradas, mientras que la sección de saveButon servirá para poder validar los datos asignados a los diferentes campos. La cual tiene añadido un mensaje que responderá en caso de que ocurra un daño.

```dart
final CancelButon = Material(
  elevation: 5.0,
  borderRadius: BorderRadius.circular(30.0),
  color: const Color(0xFFF46161),
  child: MaterialButton(
    minWidth: MediaQuery.of(context).size.width,
    padding: const EdgeInsets.fromLTRB(20.0, 15.0, 20.0, 15.0),

    onPressed: () {
      _planta_condicion.text = '';
      _planta_nombre.text = '';
      _planta_description.text= '';
      _planta_recordatorio.text ='';
      _planta_exposicion.text = '';
    },
    child: Text(
      "Borrar todo",

      style: TextStyle(
        color: Colors.white,
        fontSize: 17,
      ), // TextStyle
      textAlign: TextAlign.center,
    ), // Text
  ), // MaterialButton
); // Material
```

Botón de cancelación, borrará los datos en los campos lo que permitirá volver a ingresar datos.

```dart
return Scaffold(
  resizeToAvoidBottomInset: false,
  backgroundColor: Color(0xFFF5F5F5),
  appBar: PreferredSize(
    preferredSize: const Size.fromHeight(1),
    child: AppBar(
      backgroundColor: const Color(0xFFFCFCFC),
      title: const Text('Agregar Planta'),
      automaticallyImplyLeading: false,
        actions: const [ ],
        centerTitle: false,
        elevation: 2,
      ), // AppBar
    ), // PreferredSize
    body: SafeArea(
      child: GestureDetector(
        onTap: () => FocusScope.of(context).unfocus(),
        child: Padding(
          padding: const EdgeInsetsDirectional.fromSTEB(20, 0, 20, 0),
          child: ListView(
            padding: EdgeInsets.zero,
            scrollDirection: Axis.vertical,
```

Establecemos el Scaffold, creamos una AppBar, y agregamos SafeArea que será el widget principal.

```
Align(
  alignment: const AlignmentDirectional(0, 0),
  child: Padding(
    padding: const EdgeInsetsDirectional.fromSTEB(0, 20, 0, 0),
    child: Image.asset('assets/logo-removebg-preview.png',
      width: 120,
      height: 100,
      fit: BoxFit.cover,
    ), // Image.asset
  ), // Padding
), // Align
//titulo iniciar secion
Container(
  width: 100,
  height: 100,
  decoration: const BoxDecoration(
    color: Color(0xFFF5F5F5),
  ), // BoxDecoration
  child: const Align(
    alignment: AlignmentDirectional(0, -0.15),
    child: Text(
      'Registro de planta',
      style: TextStyle(
        fontFamily: 'Merriweather',
        color: Color.fromARGB(255, 26, 30, 26),
        fontWeight: FontWeight.w600,
        fontSize: 25,
      ), // TextStyle
    ), // Text
```

Se asigna un Aling, donde empezamos un Padding en el que podemos añadir un Image para añadir el logo, finalmente agregamos un texto decorado para así mandar a llamar su TextEdit correspondiente, esta estructura se repite para el resto de los campos creados.

```
                plantaexposicion,
                viewListbutton,
                CancelButon,
                const SizedBox(height: 15.0),
                SaveButon,
                const SizedBox(height: 15.0),
              ], // <Widget>[]
            ), // Column
          ), // Padding
        ), // Form
        Image.asset('assets/planta-log.png',
          width: 50,
          height: 500,
          fit: BoxFit.cover,
        ), // Image.asset
      ],
      ), // ListView
    ), // Padding
  ), // GestureDetector

```

Finalmente mandamos a llamar los botones para cancelar y salvar la información asignada en los diferentes campos, finalmente se agrega una imagen decorativa.

ListPage

```dart
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:prueba0/models/plant.dart';
import 'package:prueba0/page/addpage.dart';
import 'package:prueba0/page/editpage.dart';
import 'package:flutter/material.dart';

import '../services/firebase_crud.dart';

class ListPage extends StatefulWidget {
  @override
  State<StatefulWidget> createState() {
    return _ListPage();
  }
}

class _ListPage extends State<ListPage> {
  final Stream<QuerySnapshot> collectionReference = FirebaseCrud.readPlant();
  //FirebaseFirestore.instance.collection('Employee').snapshots();
```

Se manda a llamar sus correspondientes librerías y las rutas correspondientes para poder llamar las funciones necesarias, así como crear una instancia de la función de readPlanta para extraer su función.

```dart
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: const Color(0xFFFCFCFC),
      resizeToAvoidBottomInset: false,
      appBar: AppBar(
        title: const Text("Mis Plantas"),
        backgroundColor: const Color(0xD76FB244),
        actions: <Widget>[
          IconButton(
            icon: Icon(
              Icons.app_registration,
              color: Colors.white,
            ), // Icon
            onPressed: () {
              Navigator.pushAndRemoveUntil<dynamic>(
                context,
                MaterialPageRoute<dynamic>(
                  builder: (BuildContext context) => AddPage(),
                ), // MaterialPageRoute
                (route) =>
                    false, //if you want to disable back feature set to false
              );
            },
          ) // IconButton
```

Se crea y asigna una Appbar junto con un iconButton con el que podrá acceder a la página de AddPlant,

```
46          body: StreamBuilder(
47            stream: collectionReference,
48            builder: (BuildContext context, AsyncSnapshot<QuerySnapshot> snapshot) {
49              if (snapshot.hasData) {
50                return Padding(
51                  padding:
52                      const EdgeInsets.only(top: 20.0, left: 20.0, right: 20.0),
53                  child: ListView(
54                    children: snapshot.data!.docs.map((e) {
55                      return Card(
56                          margin: EdgeInsets.fromLTRB(0.0, 10.0, 0.0, 10.0),
57                          elevation: 10.0,
58                          color: Colors.white,
59                          child: Column(children: [
60                            ListTile(
61                              title: Text(
62                                e["planta_nombre"],
63                                style: TextStyle(
64                                  fontWeight: FontWeight.bold,
65                                  fontSize: 25,
66                                  color: Color(0xFF2D382F)), // TextStyle
67                              ), // Text
```

La estructura presentada muestra un texto junto con la información recuperada de la base de datos.

```
69                          child: (Column(
70                            crossAxisAlignment: CrossAxisAlignment.start,
71                            children: <Widget>[
72                              Text(
73                                  "Planta descripcion: " +
74                                      e['planta_descripcion'],
75                                  style: const TextStyle(fontSize: 17, color: Color(0xFF6A7245))), // Text
76                              Text(
77                                  "Planta Condicion: " +
78                                      e['planta_condicion'],
79                                  style: const TextStyle(fontSize: 17, color: Color(0xFF6A7245))), // Text
80                              //Text("Planta Exposicion: " + e['planta_exposicion'],
81                              //style: const TextStyle(fontSize: 12)),
82                              Text(
83                                  "Planta Recordatorio: " +
84                                      e['planta_recordatorio'],
85                                  style: const TextStyle(fontSize: 17, color: Color(0xFF6A7245))), // Text
86                            ], // <Widget>[]
87                          )), // Column
88                        ), // Container
89                      ), // ListTile
```

Esta estructura fue asignada se repite para poder mostrar toda la información solicitada

```
ButtonBar(
  alignment: MainAxisAlignment.spaceBetween,
  children: <Widget>[
    TextButton(
      style: TextButton.styleFrom(
        padding: const EdgeInsets.all(10.0),
        backgroundColor: Color(0xFF9DBE8B),
        textStyle: const TextStyle(fontSize: 17),
        shape: RoundedRectangleBorder(borderRadius: BorderRadius.zero)
      ),
      child: const Text('Editar',style: TextStyle(color: Colors.white)),
      onPressed: () {
        Navigator.pushAndRemoveUntil<dynamic>(
          context,
          MaterialPageRoute<dynamic>(
            builder: (BuildContext context) => EditPage(
              plant: Plant(
                uid: e.id,
                plantanombre: e["planta_nombre"],
                //plantadescripcion: e["plant_descripcion"],
                platacondicion: e["planta_condicion"],
                plantaexposicion:
                    e["planta_exposicion"],
                recordatorio: e["planta_recordatorio"],
              ), // Plant
            ), // EditPage
          ), // MaterialPageRoute
```

Se crea la función y decoración del botón de editar con el que se podrá tomar los datos que fueron modificados y aguardados en la base de datos.